REMARKS

Prior to examination of the above-identified application please enter this preliminary amendment. No new matter has been added. Applicant respectfully requests an action of the merits.

Respectfully submitted,

PILLSBURY WINTHROP LLP

Date: July 30, 2002

By: _____
Roger R. Wise
Registration No. 31,204
Attorney for Applicant(s)

725 South Figueroa Street, Suite 2800
Los Angeles, CA 90017-5406
Telephone: (213) 488-7100
Facsimile: (213) 629-1033

APPENDIX

IN THE BACKGROUND:

Please amend the heading on page 1, line 5 as follows:

1. Technical Field [of the Invention]

Please amend the paragraph beginning with "In addition, with such an advancement" on page 2, line 6 as follows:

In addition, with such an advancement in the field, users who develop applications for a single network processor will be able to upgrade to using several processors without having to rewrite any of the object management system messaging code. This approach is not limited to Intel® architectures; it may also be implemented on other non-Intel® related architectures.

IN THE BRIEF DESCRIPTION OF THE DRAWINGS:

In the brief description of FIG. 3A and FIG. 3B beginning on page 2, line 18, please amend as follows:

FIG. 3A and FIG. 3B together form a flowchart showing the [steps] operations involved in a local write lock request, according to an embodiment of the present invention.

In the brief description of FIG. 4 beginning on page 2, line 20, please amend as follows:

FIG. 4 is a flowchart showing the [steps] operations involved in a local read lock request, according to an embodiment of the present invention.

In the brief description of FIG. 5 beginning on page 2, line 22, please amend as follows:

FIG. 5 is a flowchart depicting the [steps] operations involved in a remote write lock request, according to an embodiment of the present invention.

In the brief description of FIG. 6 beginning on page 3, line 1, please amend as follows:

FIG. 6 is a flowchart showing the [steps] operations involved in a local write lock release, according to an embodiment of the present invention.

In the brief description of FIG. 7 beginning on page 3, line 3, please amend as follows:

FIG. 7 is a flowchart showing the [steps] operations involved in a remote write lock release, according to an embodiment of the present invention.

IN THE SPECIFICATION:

Please amend the paragraph beginning with "Figure 3 shows the steps involved in making" on page 7, line 5, as follows:

Figure 3 shows the [steps] operations involved in making a local write lock **210** request. If a write request is already pending, as questioned in [step] operation **300**, the system must wait until its pending status drops, as depicted in [step] operation **305**, and then increment the pending semaphore and continue, as shown in [step] operation **310**. That is, the semaphore operates as a variable with a value that indicates the status of common resource. It locks the resource that is being used. The process needing the resource checks the semaphore to determine the resource's status and then decides how to proceed. As described in [step] operation **315**, if the local module already has the write lock **210**, then the system increments a local write lock **210** count. If the write lock **210** is set by another object management system, the system must wait, as illustrated

in [step] operation **320**, for it to be released. If an object management system has just released its local lock and is in the process of sending the release to remote object management systems, as shown in [step] operation **325**, the system must wait until the release option is complete. A clear-pending lock is employed here. Next, as depicted in [step] operation **330**, the lock contention mutex is locked and a random number is created for resolving write lock **210** contention. As shown in [step] operation **335**, the system then checks a variable dedicated to write lock **210** arbitration counting. If the value is greater than zero, a request has been received by a remote object management between [steps] operations **320** and **325**. As such, the lock contention mutex is released and the flow returns to [step] operation **320**. Otherwise, the arbitration count is incremented, as illustrated in [step] operation **350**. As described in [step] operation **355**, the lock's **210** arbitration identification is set to the local object management system identification. Its priority is also set through the generation of a bound random number. The lock contention mutex is then released, as shown in [step] operation **360**. For each connection returned through an iterator provided by the list controller **130**, a write lock **210** request is made to the remote reader and writer's lock, as illustrated in [step] operation **365**. [Step] Operation **370** then examines whether the request failed. If a request fails because of contention in a remote module, the local write lock count is decremented, as shown in [step] operation **375**, and the process repeats from [step] operation **320,** as shown in [step] operation **380.** Instead of creating a new random number for resolving write lock **210** contention, the request' s priority is bumped beyond the upper boundary of the random number range. This guarantees that requests re-entering arbitration are afforded higher priority than new requests. The size of the random number generation range must be low enough to allow the arbitration to bump several times without integer overflow; enough to accommodate the maximum size of the

pending request queue. If the request did not fail, as depicted in [step] operation **385**, when all remote modules have provided the write lock **210** to the requesting module, the process returns without setting the write lock **210** in the lock component. This allows the local module to grant read locks **200**.

Please amend the paragraph beginning with "Figure 4 shows the steps involved in making a local" on page 8, line 15, as follows:

Figure 4 shows the [steps] operations involved in making a local read lock **200** request. This function does not result in any request being passed over the interface definition language interface **110**. The algorithm is as follows. As shown in [step] operation **410**, the request is sent to the local lock component. [Step] Operation **420** then examines whether a remote module owns the write lock **210**. If a remote module does not own the write lock **210**, the lock is granted, as shown in [step] operation **430**. If a remote module owns the write lock **210**, as depicted in [step] operation **440**, the write lock **210** is blocked. [Step] Operation **450** shows that the request is then retried. If the available time for the request expires, as illustrated in [step] operation **460**, the process repeats until the maximum number of retries is reached.

Please amend the paragraph beginning with "Figure 5 shows the steps involved in making a remote" on page 8, line 23, as follows:

Figure 5 shows the [steps] operations involved in making a remote write lock **210** request. This request is received over the interface definition language interface **110**. Each lock module is responsible for ensuring that it never sends a request to remote object management systems if it already owns the write lock **210**. Therefore, more than one call to this function

without an intervening release of the write lock **210** will only occur during write lock **210** contention. This allows the ownership of the write lock **210** to be changed during the write contention interval, where the write contention interval is, the time it takes for the requesting module to be granted the write lock **210** by all other modules.

Please amend the paragraph beginning with "As shown in step **510**, the lock contention" on page 9, line 8, as follows:

As shown in [step] operation **510**, the lock contention mutex is locked, and the arbitration count is checked. [Step] Operation **520** examines whether this value is greater than zero. If the value is not greater than zero, the count is incremented, as illustrated in [step] operation **530**. If the value is greater than zero, a request has been previously received. The contention for the write lock **210** must thus be resolved, as shown in [step] operation **540**. The resolve write lock contention function is called when a remote object management system requests a write lock **210**, yet the write lock **210** contention variables, priority and identification, have either been set by a local request or by a previous invocation of the write lock **210** arbitration function.

Please amend the paragraph beginning with "Step **550** examines whether the remote object management system" on page 10, line 5, as follows:

[Step] Operation **550** examines whether the remote object management system prevailed in arbitration. As shown in [step] operation **560**, if the remote object management system loses arbitration, the lock contention mutex is released and information about the arbitration winner to the requesting object management system is returned. Otherwise, as illustrated in [step] operation **570**, the request is sent to the lock component, which will block until all outstanding

read locks **200** are released. The lock contention mutex is then released, as depicted in [step] operation **580**. The [steps] operations outlined above cannot be executed in one function because remote requests are received in the interface definition language skeleton code, which must not block for an indeterminate period. Therefore, the incoming request must be asynchronous to the reply.

Please amend the paragraph beginning with "Figure 6 shows the steps involved in a local write lock" on page 10, line 14, as follows:

Figure 6 shows the [steps] operations involved in a local write lock **210** release. As shown in [step] operation **610**, a clear pending lock is set to prevent the local object management system from processing another request until the current lock has been released from all remote object management systems. The write lock **210** contention mutex is then locked and the write lock **210** count is decremented, as illustrated in [step] operation **620**. [Step] Operation **630** then examines whether the write lock **210** count is zero. If the write lock **210** count is zero, as shown in [step] operation **640**, the write lock **210** contention variables, identification and priority, must be cleared. Otherwise, some other module on the local object management system has a write lock **210**. In that case, the lock contention mutex must be released, as depicted in [step] operation **650**. [Step] Operation **660** describes the calling of the write lock **210** release functions on remote modules over the interface definition language interface **110**. This is achieved in two stages—one to clear the arbitration variables in the remote object management systems and one to enable arbitration for the next request. Otherwise, in a system of three or more object management systems, a remote object management system could arbitrate against stale arbitration values. The clear pending lock is then released, as shown in [step] operation **670**.

Please amend the paragraph beginning with "Figure 7 shows the steps involved in a remote write lock" on page 11, line 20 as follows:

Figure 7 shows the [steps] operations involved in a remote write lock **210** release. As explained in the preceding section, this activity occurs in two stages. First, as described in [step] operation **710**, the clear pending lock is set to prevent the local object management system from processing another request until the current lock has been released from all remote object management systems. The write lock **210** contention mutex is then locked and the write lock **210** contentions variables, write identification and priority, are cleared, as shown in [step] operation **720**. The writer's lock **210** is then cleared from the local lock component, as illustrated in [step] operation **730**. The lock contention mutex is released, as described in [step] operation **740**. In the second stage, the clear pending lock is cleared, as shown in [step] operation **750**. The second stage occurs after the owner of the write lock **210** on all connected object management systems has performed the first stage.

Please amend the paragraph beginning with "While the above description refers" on page 12, line 7 as follows:

While the above description refers to particular embodiments of the present invention, it will be understood to those of ordinary skill in the art that modifications may be made without departing from the spirit thereof. The accompanying claims are intended to cover any such modifications as would fall within the true scope and spirit of the embodiments of the present invention.

Please amend the paragraph beginning with "The presently disclosed embodiments" on page 12, line 11 as follows:

The presently disclosed embodiments are therefore to be considered in all respects as illustrative and not restrictive; the scope of the embodiments of the invention being indicated by the appended claims, rather than the foregoing description. All changes that come within the meaning and range of equivalency of the claims are therefore intended to be embraced therein.

IN THE CLAIMS:

Please amend claims 27 through 36 as follows:

27. (Amended) [A system for a local write lock request comprising a computer readable medium and a computer readable program code stored on the computer readable medium having instructions to:] An article comprising:

a storage medium having stored thereon instructions that when executed by a machine result in the following:

[wait] waiting for a drop in pending status of a write lock, if a write request is pending;

[augment] augmenting a pending semaphore;

[augment] augmenting a local write lock count, if a local module has the write lock;

[lock] locking a lock contention mutex and [create] creating a random number for resolving write lock contention;

[check] checking a variable dedicated to write lock arbitration counting;

[augment] augmenting the arbitration count;

[set] setting the write lock's arbitration identification and [set] setting the write lock's priority by generating a bound random number;

[release] releasing the lock contention mutex; and

[request] requesting a write lock.

28. (Amended) The [system] article of claim 27, wherein the write lock's arbitration identification is set to an identification of a local object management system.

29. (Amended) [A system for a local read lock request comprising a computer readable medium and a computer readable program code stored on the computer readable medium having instructions to:] An article comprising:

a storage medium having stored thereon instructions that when executed by a machine result in the following:

[pass] passing a local read lock request to a local lock component;

[award] awarding or [deny] denying the local read lock request;

[retry] retrying if the local read lock request is denied; and

[repeat] repeating until a maximum number of retries is reached.

30. (Amended) The [system] article of claim 29, wherein the local read lock request is denied if a write lock is owned by a remote module.

31. (Amended) [A system for a remote write lock request comprising a computer readable medium and a computer readable program code stored on the computer readable medium having instructions to:] An article comprising:

a storage medium having stored thereon instructions that when executed by a machine

result in the following:

[lock] locking a lock contention mutex and [check] checking an arbitration count;

[resolve] resolving contention for a write lock if the arbitration count is greater than zero, or [augment] augmenting the arbitration count if the arbitration count is not greater than zero; and

[release] releasing the lock contention mutex or [forward] forwarding the request to a lock component.

32. (Amended) The [system] article of claim 31, wherein the instructions are adapted to provide to an arbiter to examine a priority value and identification value.

33. (Amended) [A system for a local write lock release comprising a computer readable medium and a computer readable program code stored on the computer readable medium having instructions to:] An article comprising:

a storage medium having stored thereon instructions that when executed by a machine result in the following:

[set] setting a clear pending lock;

[lock] locking a write lock contention mutex and [decrement] decrementing a write lock count;

[clear] clearing write lock contention variables, if the write lock count is zero;

[release] releasing the lock contention mutex;

[call] calling a write lock release function; and

[release] releasing the clear pending lock.

34. (Amended) The [system] article of claim 33, wherein the clear pending lock is set to prevent a local object management system from processing another request until the lock is released from remote object management systems.

35. (Amended) [A system for a remote write lock release comprising a computer readable medium and a computer readable program code stored on the computer readable medium having instructions to:] An article comprising:

a storage medium having instructions stored thereon that when executed result in the following:

[set] setting a clear pending lock;

[lock] locking a write lock contention mutex and [clear] clearing write lock contention variables;

[clear] clearing a writer's lock from a local lock component;

[release] releasing the write lock contention mutex; and

[clear] clearing the clear pending lock.

36. (Amended) The [system] article of claim 35, wherein the write lock contention variables are identification and priority.